ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik Computer Science

Introduction to Machine Learning

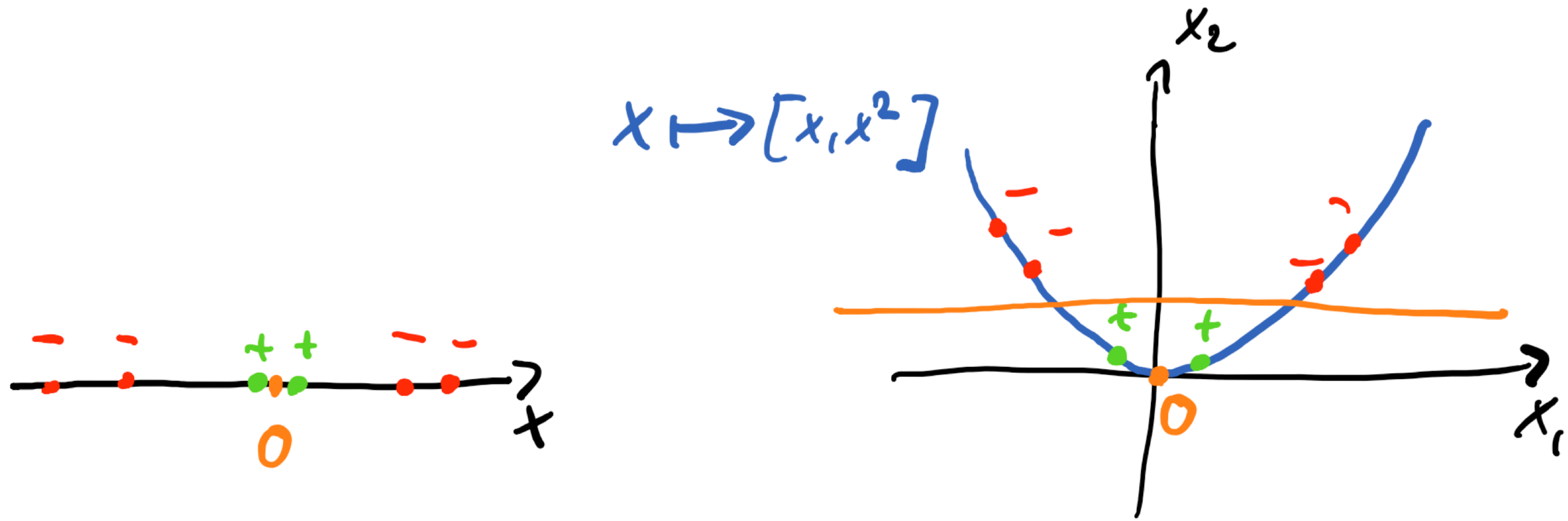
Non-linear prediction with kernels

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

Solving non-linear classification tasks

- How can we find nonlinear classification boundaries?
- Similar as in regression, can use non-linear transformations of the feature vectors, followed by linear classification

$x_1 = x,\ x_2 = x^2$



Avoiding the feature explosion

- Need $O(d^k)$ dimensions to represent (multivariate) polynomials of degree k on d features
- **Example**: $d=10000$, $k=2$ ➔ Need ~$100M$ dimensions

- In the following, we can see how we can efficiently implicitly operate in such high-dimensional feature spaces (i.e., without ever explicitly computing the transformation)

The „Kernel Trick“

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

- Example: Perceptron

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\}$$

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)\}$$

- Will see further examples later

Kernelized Perceptron

Training

- Initialize $\alpha_1 = \cdots = \alpha_n = 0$
- For *t=1,2,...*
  - Pick data point ($\mathbf{x}_i$, y_i) uniformly at random
  - Predict

$$\hat{y} = \operatorname{sign}\left(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i)\right)$$

  - If $\hat{y} \neq y_i$ set $\alpha_i \leftarrow \alpha_i + \eta_t$

Prediction

- For new point x, predict

$$\hat{y} = \operatorname{sign}\Big(\underbrace{\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})}_{w^T\phi(x)}\Big)$$

Questions

- What are valid kernels?
- How can we select a good kernel for our problem?
- Can we use kernels beyond the perceptron?
- Kernels work in very high-dimensional spaces.
  Doesn‘t this lead to overfitting?

Definition: kernel functions

- Data space *X*
- A **kernel** is a function $k : X \times X \to \mathbb{R}$ satisfying
- 1) Symmetry: For any $\mathbf{x}, \mathbf{x}' \in X$ it must hold that

  $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$

- 2) Positive semi-definiteness: For any *n*, any set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$, the kernel (Gram) matrix

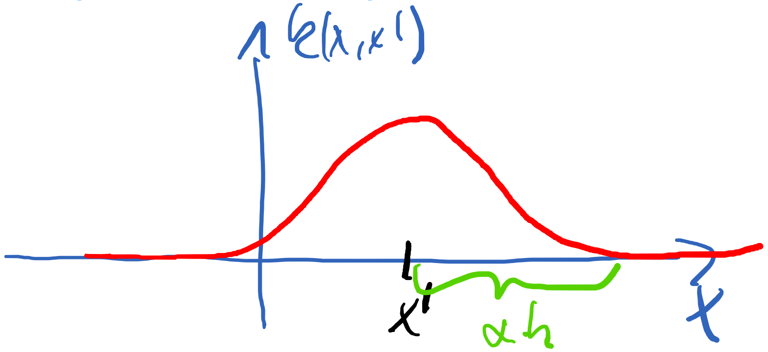
$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \ldots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

  must be positive semi-definite

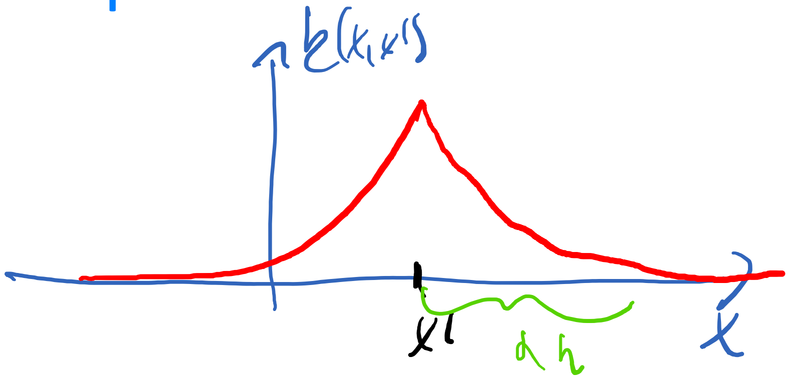
Examples of kernels on $\mathbb{R}^d$

- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$
- Gaussian (RBF, squared exp. kernel): $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2 / h^2)$

"Bandwidth"/ Length scale parameter



- Laplacian kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_1 / h)$



Examples of (non)-kernels

$$k(x, x') = \sin(x)\cos(x')$$

not symmetric: $k(x,x') \neq k(x',x)$ e.g. for $x=0$, $x' = \frac{\pi}{2}$

$\Rightarrow$ not a valid kernel on $\mathbb{R}$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T M \mathbf{x}'$$

$x, x' \in \mathbb{R}^d$, $M \in \mathbb{R}^{d \times d}$

if M symmetric: $k(x,x') = x^T M x' = x^T M^T x' = x'^T M x = k(x',x)$ ✓

if M not symm., k in general not sym.

Claim k s.p.d. $\Leftrightarrow$ M s.p.d. (sem.)

if M not pos.def.: E.g. in 1-dim: $M=-1$: $x \cdot M \cdot x = -x^2 < 0$ if $x \neq 0$

if M pos semidef.: $M = U D^{\frac{1}{2}} D^{\frac{1}{2}T} U^T = V^T V$ for $V = (U D^{\frac{1}{2}})^T$

then: $k(x,x') = x^T M x' = x^T V^T V x' = (Vx)^T (Vx') = \phi(x)^T \phi(x')$ $\square$

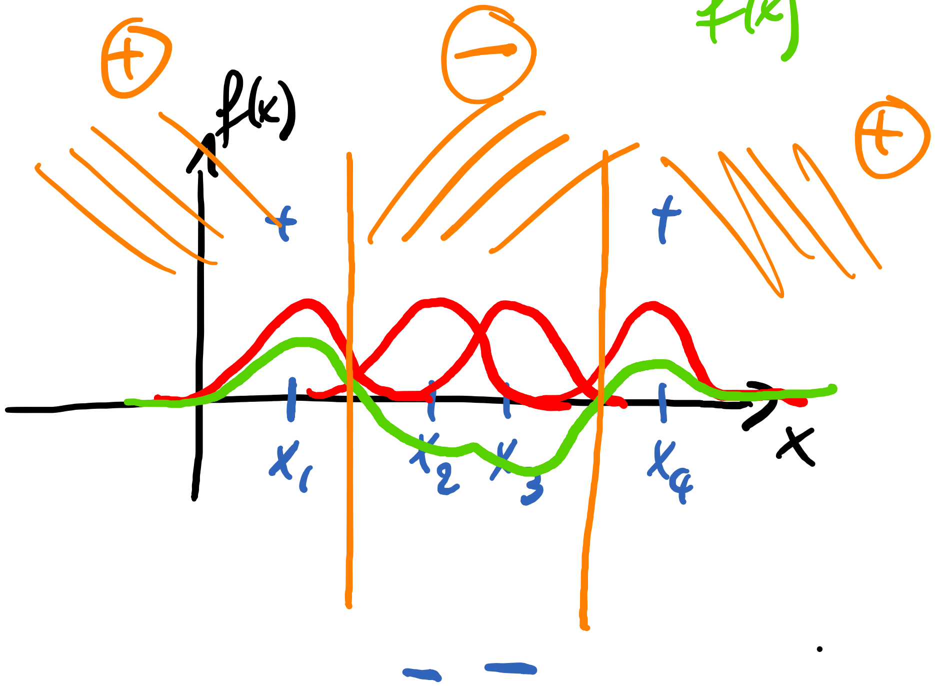
for $\phi(x) = Vx$

Effect of kernel on function class

- Given kernel k, predictors (for kernelized classification) have the form

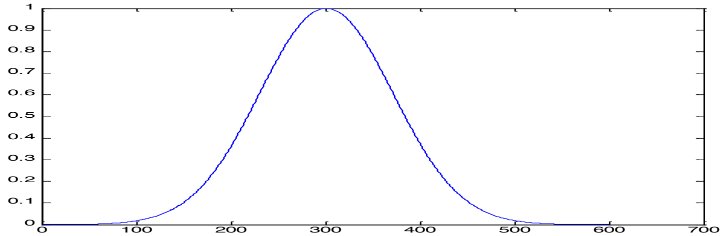
$$\hat{y} = \mathrm{sign}\left(\underbrace{\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})}_{f(x)}\right)$$

$D = \{(x_1, +), (x_2, -),$
$(x_3, -), (x_4, +)\}$

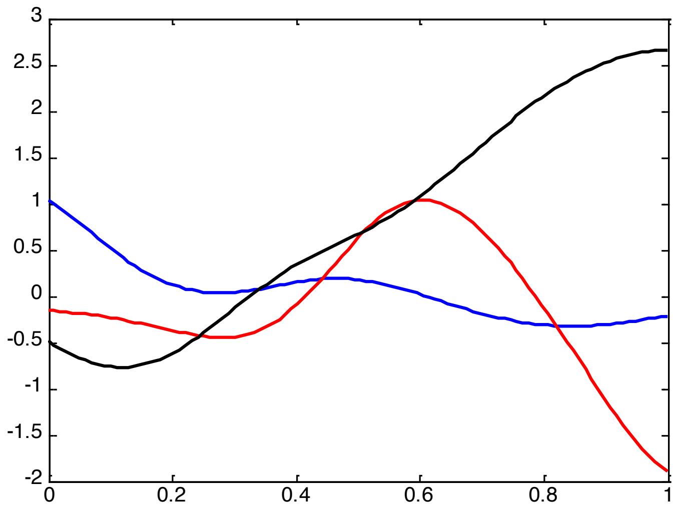


Example: Gaussian kernel

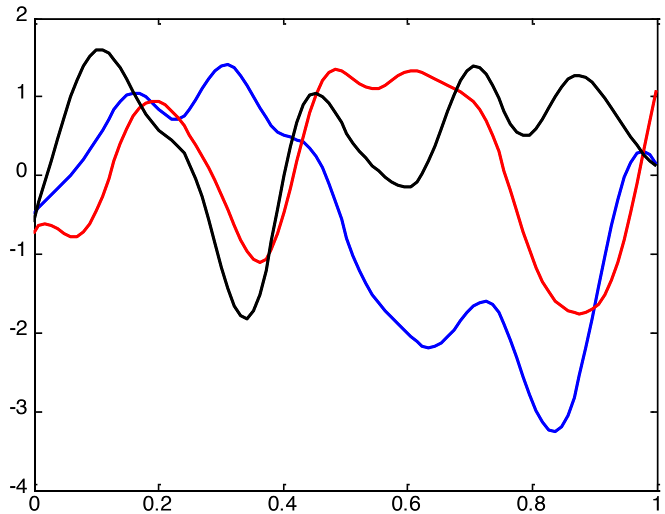
$$k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||_2^2/h^2)$$



$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$



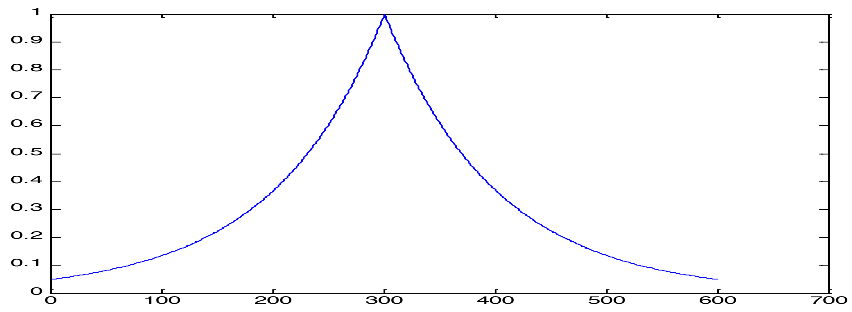
Bandwidth h=.3



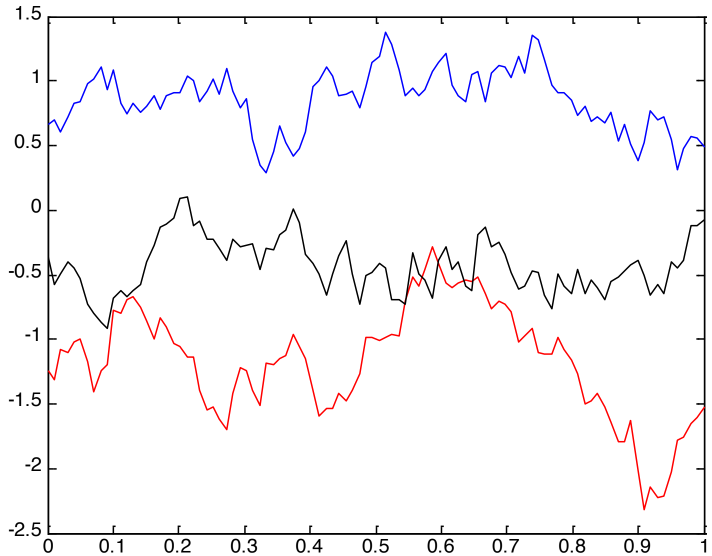
Bandwidth h=.1

Example: Laplace/Exponential kernel

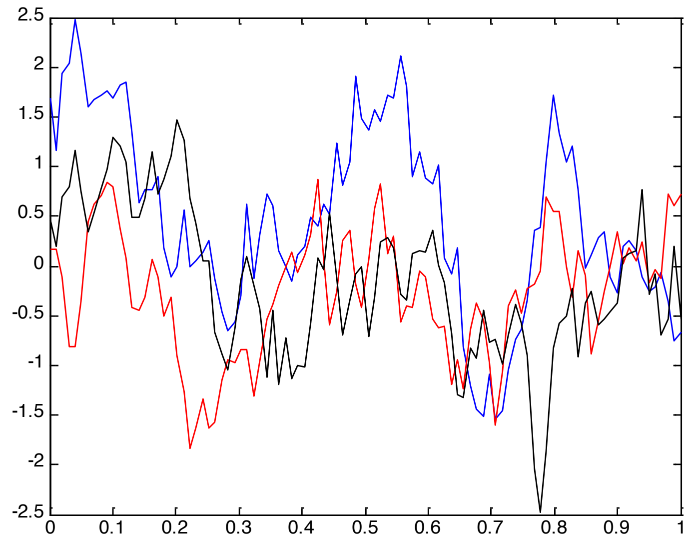
$$k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||_1/h)$$



$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$



Bandwidth h=1



Bandwidth h=.3

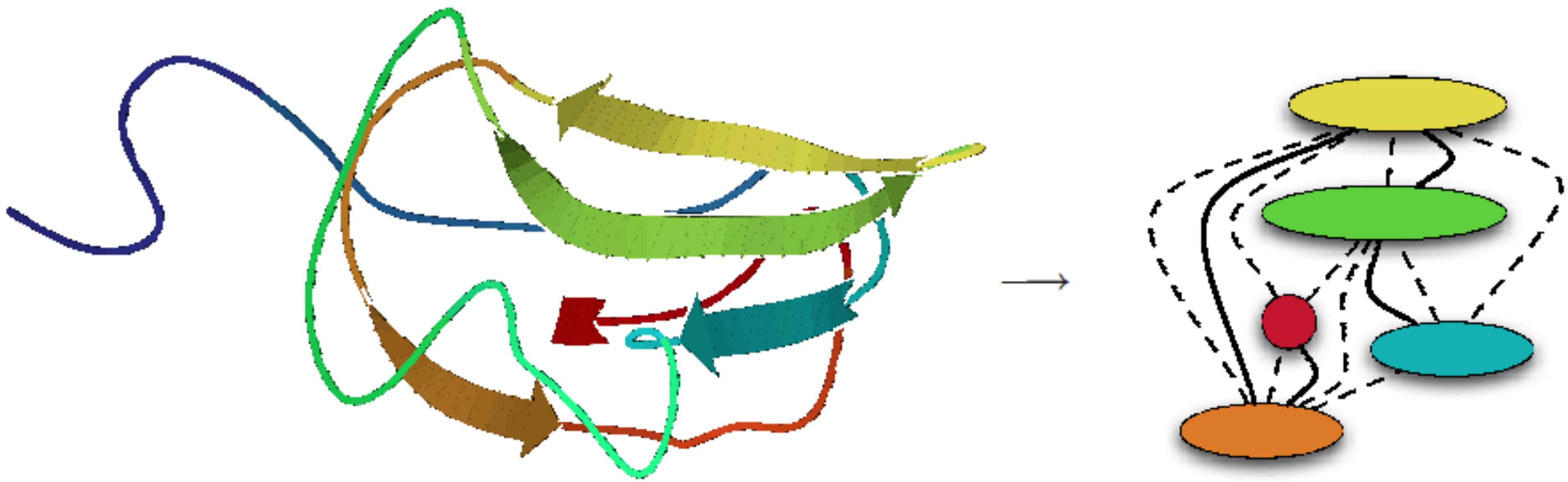
Demo: Effect on decision boundary

Kernels beyond $\mathbb{R}^d$

- Can define kernels on a variety of objects:

- Sequence kernels
- Graph kernels
- Diffusion kernels
- Kernels on probability distributions
- ...

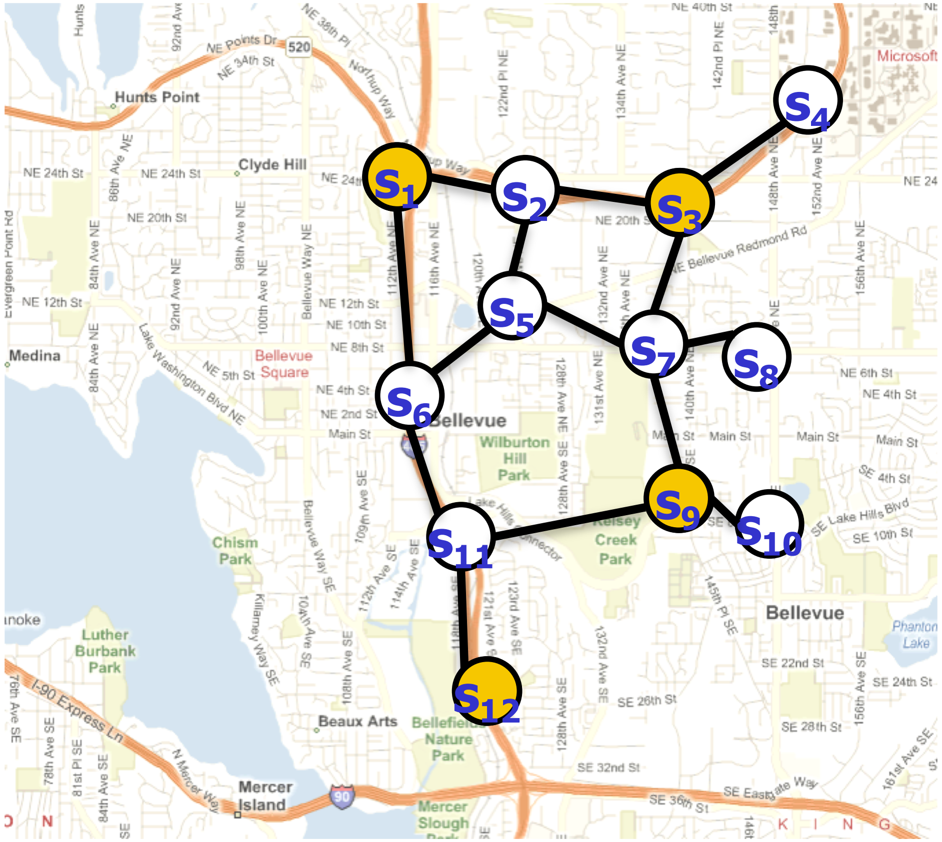
Example: Graph kernels



[Borgwardt et al.]

- Can define a kernel for measuring similarity between graphs by comparing random walks on both graphs (not further defined here)

Example: Diffusion kernels on graphs



$$\mathbf{K} = \exp(-\beta \mathbf{L})$$

- Can measure similarity among nodes in a graph via diffusion kernels (not defined here)

Kernel engineering (composition rules)

- Suppose we have two kernels

$k_1(x) = \phi(x)^T \phi(x')$

$$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad\qquad k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

defined on data space *X*

- Then the following functions are valid kernels:

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')\ k_2(\mathbf{x}, \mathbf{x}')$$
$$k(\mathbf{x}, \mathbf{x}') = c\, k_1(\mathbf{x}, \mathbf{x}') \text{ for } c > 0$$
$$k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}'))$$

Sps: $\nu : Z \to X$

$k_2(z, z') = k_1(\nu(z), \nu(z'))$ is a valid kernel:

$k_2(z, z') = \underbrace{\phi(\nu(z))}_{\psi(z)}{}^T \underbrace{\phi(\nu(z'))}_{\psi(z')}$

where *f* is a polynomial with positive coefficients or the exponential function

Example: ANOVA kernel

$$k(x, x') = \sum_{j=1}^{d} k_j(x_j, x'_j)$$

where $x, x' \in \mathbb{R}^d$

$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

$k_j : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ kernel

$\hookrightarrow$ k is a valid kernel

What functions does k model?

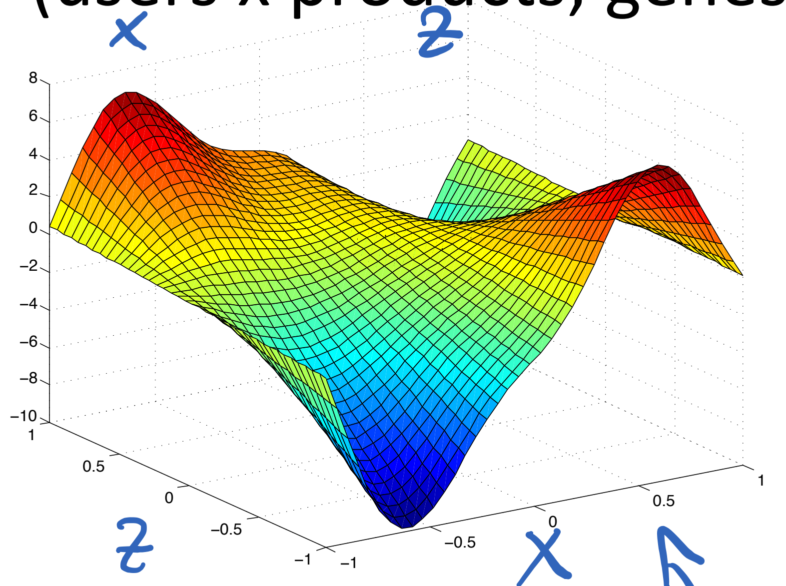
$$f(x) = \sum_{i=1}^{n} \alpha_i y_i k(x^{(i)}, x)$$

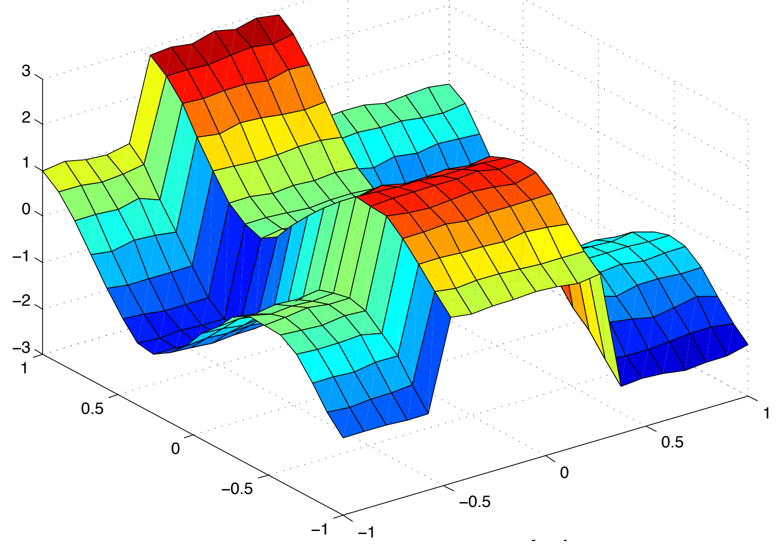
$$= \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{d} k_j(x_j^{(i)}, x_j)$$

$$= \sum_{j=1}^{d} \underbrace{\sum_{i=1}^{n} \alpha_i y_i k_j(x_j^{(i)}, x_j)}_{f_j(x_j)}$$

Example: Modeling pairwise data

- May want to use kernels to model pairwise data (users x products; genes x patients; ...)





$$k((x,z),(x',z')) = k_x(x,x') \cdot k_z(z,z')$$

$$k((x,z),(x',z')) = k_x(x,x') + k_z(z,z')$$

Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

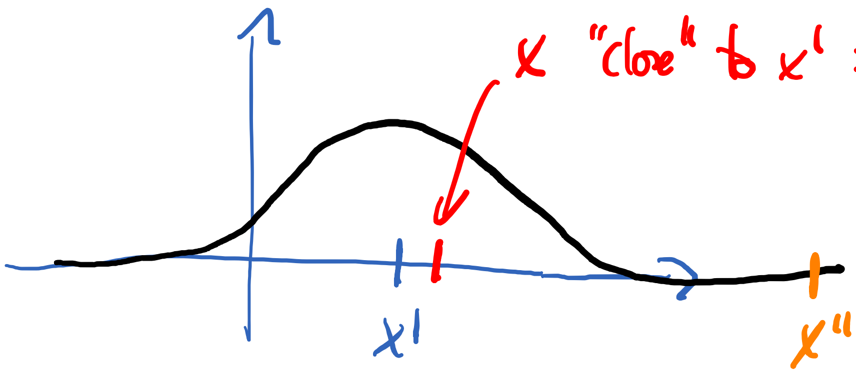
- Next questions:
  - What kind of predictors / decision boundaries do kernel methods entail?
  - Can we use the kernel trick beyond the perceptron?

Kernels as *similarity functions*

- Recall Perceptron (and SVM) classification rule:

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right) \approx \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i [x_i \text{ "close" to } x]\right)$$

- Consider Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||^2/h^2)$



Side note: Nearest-neighbor classifiers

- For data point ***x***, predict majority of labels of *k* nearest neighbors

$$y = \operatorname{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors of } \mathbf{x}] \right)$$

Demo: k-NN

Nearest-neighbor classifiers

- For data point ***x***, predict majority of labels of *k* nearest neighbors

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors of } \mathbf{x}]\right)$$

- How to choose *k*?
  - Cross-validation! ☺

K-NN vs. Kernel Perceptron

- k-Nearest Neighbor:

$$y = \mathrm{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors of } \mathbf{x}]\right)$$

- Kernel Perceptron:

$$y = \mathrm{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Comparison: k-NN vs Kernelized Perceptron

Method	*k-NN*	*Kernelized Perceptron*
Advantages	No training necessary	Optimized weights can lead to improved performance Can capture „global trends“ with suitable kernels Depends on „wrongly classified“ examples only
Disadvantages	Depends on all data ➔ inefficient	Training requires optimization

Parametric vs nonparametric learning

- Parametric models have finite set of parameters
- **Example**: Linear regression, linear Perceptron, ...

- Nonparametric models grow in complexity with the size of the data
  - Potentially much more expressive
  - But also more computationally complex – Why?
- **Example**: Kernelized Perceptron, k-NN, ...

- Kernels provide a principled way of deriving non-parametric models from parametric ones

Where are we?

- We've seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

- Next question:
  - Can we use the kernel trick beyond the perceptron?

Kernelized SVM

- The support vector machine

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda ||\mathbf{w}||_2^2$$

can also be kernelized

How to kernelize the objective?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \underbrace{\max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\}}_{(*)} + \lambda ||\mathbf{w}||_2^2$$

$$\boxed{w = \sum_{j=1}^{n} \alpha_j y_j x_j}$$

$$(*) = \max\{0, 1 - y_i \left(\sum_{j=1}^{n} \alpha_j y_j x_j\right)^T x_i\}$$

$$= \max\{0, 1 - y_i \sum_j \alpha_j y_j \underbrace{(x_j^T x_i)}_{k(x_i, x_j)}$$

$$= \max\{0, 1 - y_i \alpha^T k_i\}, \quad k_i = [y_1 k(x_i, x_1) \ldots y_n k(x_i, x_n)]^T$$

$$\alpha = [\alpha_1 \ldots \alpha_n]^T$$

How to kernelize the regularizer?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \underbrace{\lambda ||\mathbf{w}||_2^2}_{(*)}$$

$$(*) = \lambda \cdot w^T w = \lambda \left(\sum_{i=1}^{n} \alpha_i y_i x_i\right)^T \left(\sum_{j=1}^{n} \alpha_j y_j x_j\right)$$

$$= \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{x_i^T x_j}_{k(x_i, x_j)}$$

$$= \lambda \cdot \alpha^T D_y K D_y \alpha$$

$$D_y = \begin{pmatrix} y_1 & & 0 \\ & \ddots & \\ 0 & & y_n \end{pmatrix}$$

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

Learning & prediction with kernel classifier

- Learning: Solve the problem

Perceptron:

$$\arg\min_{\alpha} \frac{1}{n}\sum_{i=1}^{n} \max\{0, -y_i\alpha^T\mathbf{k}_i\}$$

Or:

SVM:

$$\arg\min_{\alpha} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\alpha^T\mathbf{k}_i\} + \lambda\alpha^T\mathbf{D_y}\mathbf{K}\mathbf{D_y}\alpha$$

$$\mathbf{k}_i = [y_1 k(\mathbf{x}_i, \mathbf{x}_1), \ldots, y_n k(\mathbf{x}_i, \mathbf{x}_n)]$$

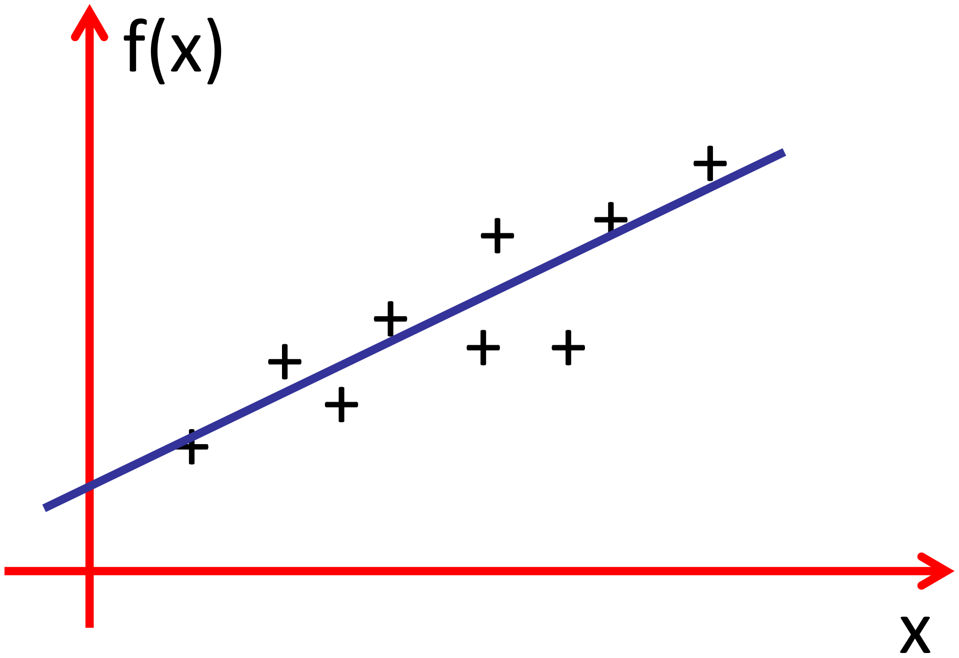
- Prediction: For data point ***x*** predict label *y* as

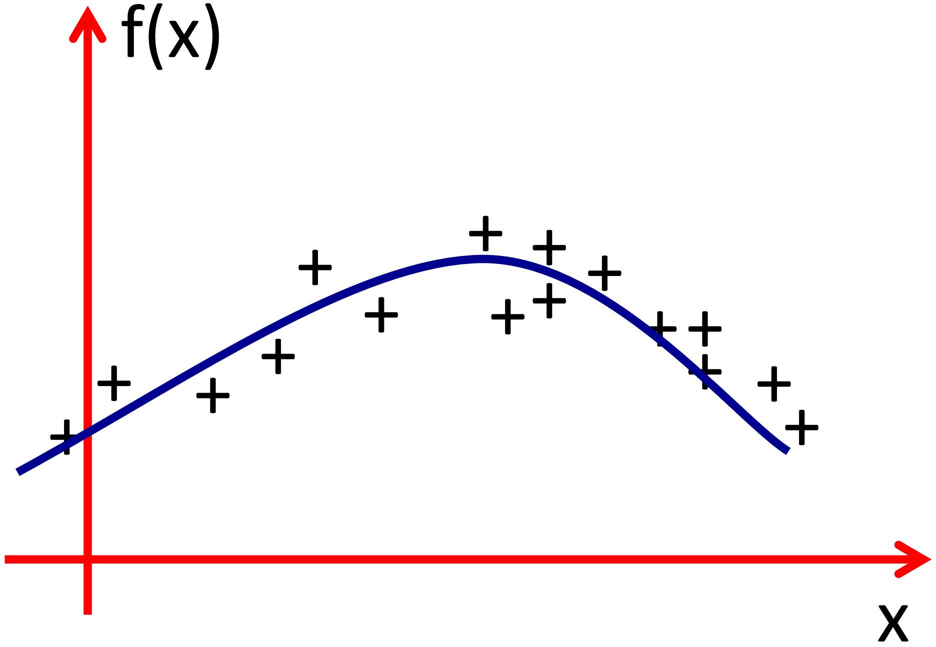
$$\hat{y} = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Demo: Kernelized SVM

Kernelized Linear Regression

- From linear to nonlinear regression:





- Can also kernelize linear regression
- Predictor has the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

Example: Kernelized linear regression

- Original (parametric) linear optimization problem

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}_i - y_i \right)^2 + \lambda ||\mathbf{w}||_2^2$$

- Similar as in perceptron, optimal   lies in span of data:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \underline{\alpha_i} \mathbf{x}_i$$

Kernelizing linear regression

$$\underbrace{\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2}_{(***)} \quad \underbrace{\left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2}_{(*)} + \underbrace{\lambda||\mathbf{w}||_2^2}_{(**)}$$

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

$$(*) = \left(\left(\sum_{j=1}^{n} \alpha_j x_j\right)^T x_i - y_i\right)^2$$

$$= \left(\sum_{j=1}^{n} \alpha_j \underbrace{(x_j^T x_i)}_{k(x_i, x_j)} - y_i\right)^2$$

$$= \left(\alpha^T k_i - y_i\right)^2$$

$$K = \left[k_1 | \cdots | k_n\right]$$

$$k_i = \begin{pmatrix} k(x_i, x_1) \\ \vdots \\ k(x_i, x_n) \end{pmatrix}$$

$$(**) = \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j \underbrace{x_i^T x_j}_{k(x_i, x_j)}$$

$$= \alpha^T K \alpha$$

$$(***): \quad \hat{\alpha} = \arg\min_{\alpha \in \mathbb{R}^n} \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left(\alpha^T k_i - y_i\right)^2}_{||\alpha^T K - y||_2^2} + \lambda \alpha^T K \alpha$$

Learning & Predicting with KLR

- Learning: Solve least squares problem

$$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} ||\alpha^T \mathbf{K} - \mathbf{y}||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Closed-form solution: $\hat{\alpha} = (\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{y}$

- Prediction: For data point ***x*** predict response *y* as

$$\hat{y} = \sum_{i=1}^{n} \hat{\alpha}_i k(\mathbf{x}_i, \mathbf{x})$$